В этом посте затроним тему стандартных потоков ввода/вывода. Вначале ответим на вопросы:

1. Что это за потоки
2. Для чего они нужны?

Я думаю, вы прекрасно понимаете, что программы в большинстве случаев выводят результаты своей работы на экран, а считывают информацию с клавиатуры. **Так вот, на уровне операционной системы, как правило, имеются три стандартных потока ввода/вывода**:

- stdout - поток вывода информации
- stderr - поток выдачи ошибок
- stdin - поток ввода информации

Первые два, как правило, пользуются монитором, а третий - клавиатурой, но все эти потоки можно настроить на любые другие устройства (кто-то видел, как парень проходил Dark Souls используя бананы?). **Благодаря использованию стандартных потоков ввода/вывода, программы способны универсальным образом работать с любыми устройствами, связанными с этими потоками**. И это очень удобно

На программном уровне все эти потоки организованы в виде буферов приема или передачи информации. То есть, данные сначала поступают в буфер, а затем, уже либо на устройство вывода, либо в переменные программы. Это очень важный момент. Данные поступают в программу, например, с клавиатуры, не напрямую, а через буфер ввода. То же самое с выводом. Сначала данные из программы попадаю в буфер вывода и только потом, например, отображаются на мониторе или записываются в файл. Запомним этот момент. Он нам в будущем пригодится.

Функции для работы со стандартными потоками

Язык Си предоставляет набор библиотечных функций для работы со стандартными потоками ввода/вывода. Вот примеры некоторых часто используемых из них:

- putchar() – вывод символа через поток stdout;
- printf() – форматный вывод строки через поток stdout;
- perror() – вывод ошибок в виде строки через поток stderr;

- getchar() – чтение одного байта (символа) из потока stdin;
- scanf() – форматный ввод данных из потока stdin.

Описания (прототипы) всех этих функций даны в заголовочном файле **stdio**.****h**. То есть, для их использования в программе вначале должна быть прописана директива:

```language-c
#include <stdio.h>
```

Функция getchar()

Давайте рассмотрим эти функции и начнём с getchar(). Она имеет следующее определение:

int getchar(void)

Целочисленный тип int перед функцией означает, что она возвращает целое число, а void в круглых скобках говорит об отсутствии каких-либо параметров. Поэтому мы можем вызвать её в программе следующим образом:

```language-c
#include <stdio.h>

int main(){
        int value = getchar();
        return 0;
}
```

После запуска этой программы **будет ожидаться ввод значения в поток stdin**. Так как он по умолчанию ассоциирован с клавиатурой, то нам нужно набрать один символ, любой, например, буквы ‘s’ и нажать клавишу Enter. После этого программа продолжится, перейдет к следующему оператору return и завершится.

Я напомню, что функция getchar() предназначена для чтения одного байта из потока stdin. Тогда возникает вопрос, почему эта функция возвращает целое число типа *int*, а не данные типа *char*, который и описывает один байт памяти? Дело в том, что stdin работает несколько более сложным образом, нежели просто выдача очередного байта из буфера. В частности, он дополнительно генерирует некоторые служебные значения. Например, значение:

EOF (End of File - конец файла)

Которое определено, ка к-1 в заголовочном файле stdio.h. То есть помимо байтового диапазона [0; 255] целых чисел функция getchar() дополнительно может вернуть значения вне этого диапазона, в частности, -1.

Вам может показаться странным, что мы говорим про константу EOF, когда речь идет о вводе данных с клавиатуры? Но, во-первых, стандартный поток ввода stdin вполне можно связать с файлом и тогда данные будут читаться из него, а не с клавиатуры и при достижении конца файла будет сгенерировано значение EOF. И, во-вторых, при вводе с клавиатуры мы также можем сымитировать достижение конца файла путем ввода специального символа комбинацией клавиш Ctrl+Z для ОС Windows и Ctrl+D для ОС Linux.

Таким образом, функции getchar() нужно возвращать целые значения, превышающий байтовый диапазон [0; 255]. Поэтому разработчик языка Си решил использовать тип int.

Функция putchar()

Следующая аналогичная функция – это putchar(), которая служит для вывода одного байта (символа) в выходной поток stdout и определена следующим образом:

```
int putchar(int ch);
```

Она также возвращает целое число типа int и в качестве аргумента принимает целое значение этого же типа int. В действительности тип int здесь использован для сопряжения (по типам данных) с функцией getchar(). Иначе бы можно было прописать тип char, так как функция putchar() в качестве аргумента принимает код символа в диапазоне [0;255]. Любое другое значение за пределами этого диапазона просто будет приводиться к восьми битам и затем помещаться в выходной поток stdout. Возвращает эта функция код символа, переданного в выходной поток:

```c
#include <stdio.h>

int main(void){
        int value = getchar();
        int res = putchar(value);
        printf("\n%d\n", res);
        return 0;
}
```

При выполнении этой программы, нам необходимо будет ввести какой-либо символ с клавиатуры, и затем, он продублируется вызовом функции putchar(). Возвращаемое значение (код введенного символа) будет выведено на экран с помощью функции printf

Конечно, на практике функцию putchar() **обычно вызывают исключительно для вывода информации в стандартный поток stdout**. Поэтому возвращаемое значение просто игнорируют:

```c
putchar(value);
```

То есть, если функция что-либо возвращает, нет необходимости в программе учитывать это значение. В этом случае говорят, что **функция вызвана ради побочного эффекта**. Надо сказать, что в языке Си это обычная практика.

Буферы приема/передачи стандартных потоков

Я акцентирую ваше внимание на наличие буферов приёма/передачи информации у стандартных потоков ввода/вывода. При запуске программы они пустые, в них нет никаких посторонних значений. Но, в процессе ввода или вывода информации они заполняются и это может повлиять на ход исполнения программы.

Давайте я это покажу на конкретном примере. Запишем два подряд идущих вызова функции getchar() следующим образом:

```c
#include <stdio.h>

int main(void){
        int value1 = getchar();
        int value2 = getchar();
        printf("%c %c\n", value1, value2);
        return 0;
}
```

И после запуска этой программы введём два символа: nt (можете сами ввести любую другую комбинацию). В результате оба символа помещаются во входной буфер, первый считывается при вызове функции getchar(), а второй символ — при втором вызове. Поэтому программа не ждёт от нас ввода какой-либо дополнительной информации, а сразу переходит к printf(). Соответственно в переменной value1 будет храниться код символа n, а в

переменной value2 – код символа t. Затем, функция printf() выводит на экран оба прочитанных символа.

Вот наглядный эффект работы входного буфера. Мало того, если бы мы ввели не два, а, скажем, три символа, то после чтения первых двух, последний так бы и остался во входном буфере до момента завершения программы. При завершении, все буферы автоматически очищаются.